

1 Problem 1

1.1 Code

```
1 import java.util.HashMap;
2
3 /* CT255 Assignment 2
4  * This class provides functionality to build rainbow tables (with a different reduction function
5   per round) for 8 character long strings, which
6   consist of the symbols "a .. z", "A .. Z", "0 .. 9", "!" and "#" (64 symbols in total).
7   Properly used, it creates the following value pairs (start value - end value) after 10,000
8   iterations of hashFunction() and reductionFunction():
9
10    start value - end value
11    Kermit12    1sXcRAuN
12    Modulus!    L2rEsY8h
13    Pigtail1    R0NoLfOw
14    GalwayNo    9PZjwF5c
15    Trumpets    !oeHRZpK
16    HelloPat    dkMPG7!U
17    pinky##!    eDx58HRq
18    01!19!56    vJ90ePjV
19    aaaaaaaa    rLtVvpQS
20    aaaaaaaa    klQ6IeQJ
21
22  */
23
24 public class RainbowTable
25 {
26     public static void main(String[] args) {
27         long res = 0;
28
29         // String array of the known passwords
30         String[] passwords = {"Kermit12", "Modulus!", "Pigtail1", "GalwayNo", "Trumpets", "HelloPat",
31                                "pinky##!", "01!19!56", "aaaaaaa", "aaaaaaa"};
32
33         HashMap<String, String> rainbowTable = new HashMap<>(); // declaring a HashTable that i'll
34                                use to store the password : hash pairs
35
36         // looping through the passwords array
37         for (String start : passwords) {
38             if (start.length() != 8) {
39                 System.out.println("Input " + start + " must be 8 characters long - Exit");
40             }
41             else {
42                 String hash = start; // declaring a String hash that
43                                     will hold the final reduced hash of a given password
44
45                 // hashing & reducing the start String 10000 times.
46                 for (int i = 0; i < 10000; i++) {
47                     hash = reductionFunction((hashFunction(hash)), i);
48                 }
49
50                 // adding the password & its hash value to the rainbowTable HashMap
51                 rainbowTable.put(start, hash);
52             }
53         }
54         // printing out the contents of the rainbowTable
55         System.out.println(rainbowTable);
56
57     }
58
59     private static long hashFunction(String s){
60         long ret = 0;
61         int i;
62         long[] hashA = new long[]{1, 1, 1, 1};
63
64         String filler, sIn;
65
66         int DIV = 65536;
67
68         filler = new String("ABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGH");
69
70         sIn = s + filler; // Add characters, now have "<input>ABCDEFGH..."
71         sIn = sIn.substring(0, 64); // Limit string to first 64 characters
72
73         for (i = 0; i < sIn.length(); i++) {
74             char byPos = sIn.charAt(i); // get i'th character
```

```

72         hashA[0] += (byPos * 17111); // Note: A += B means A = A + B
73         hashA[1] += (hashA[0] + byPos * 31349);
74         hashA[2] += (hashA[1] - byPos * 101302);
75         hashA[3] += (byPos * 79001);
76     }
77
78     ret = (hashA[0] + hashA[2]) + (hashA[1] * hashA[3]);
79     if (ret < 0) ret *= -1;
80     return ret;
81 }
82
83 private static String reductionFunction(long val, int round) { // Note that for the first
84     function call "round" has to be 0, // and has to be incremented by
85     String car, out; // one with every subsequent call. // I.e. "round" created
86     int i; // variations of the reduction function.
87     char dat;
88     car = new String("0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ!#");
89     out = new String("");
90
91     for (i = 0; i < 8; i++) {
92         val -= round;
93         dat = (char) (val % 63);
94         val = val / 83;
95         out = out + car.charAt(dat);
96     }
97
98     return out;
99 }
100 }

```

1.2 Output

```

[andrew@inspiron3501 CT255-Assignment-2]$ javac RainbowTable.java && java RainbowTable
{Kermit12=lsXcRAuN, GalwayNo=9PZjwF5c, aaaaaaaa=rLtVvpQS, HelloPat=dkMPG7!U, Modulus!=L2rEsY8h, Pigtail1=RONoLfOw, pinky##!=eDx58HRq, 01!19!56=vJ90ePjV, Trumpets=!oeHRZpK}
[andrew@inspiron3501 CT255-Assignment-2]$

```

2 Problem 2

2.1 Code

```

1  import java.util.HashMap;
2
3  /* CT255 Assignment 2
4   * This class provides functionality to build rainbow tables (with a different reduction function
5   * per round) for 8 character long strings, which
6   * consist of the symbols "a .. z", "A .. Z", "0 .. 9", "!" and "#" (64 symbols in total).
7   * Properly used, it creates the following value pairs (start value - end value) after 10,000
8   * iterations of hashFunction() and reductionFunction():
9   *
10   * start value - end value
11   * Kermit12 - lsXcRAuN
12   * Modulus! - L2rEsY8h
13   * Pigtail1 - RONoLfOw
14   * GalwayNo - 9PZjwF5c
15   * Trumpets - !oeHRZpK
16   * HelloPat - dkMPG7!U
17   * pinky##! - eDx58HRq
18   * 01!19!56 - vJ90ePjV
19   * aaaaaaaa - rLtVvpQS
20   * aaaaaaaa - klQ6IeQJ
21
22   *
23   * @author Michael Schukat
24   * @version 1.0
25   */
26 public class RainbowTable
27 {
28     public static void main(String[] args) {
29         long res = 0;
30
31         // String array of the known passwords
32         String[] passwords = {"Kermit12", "Modulus!", "Pigtail1", "GalwayNo", "Trumpets", "HelloPat",
33             "pinky##!", "01!19!56", "aaaaaaa", "aaaaaaa"};
34
35         HashMap<String, String> rainbowTable = new HashMap<>(); // declaring a HashTable that i'll
36             use to store the password : hash pairs
37
38         // looping through the passwords array
39         for (String start : passwords) {
40             if (start.length() != 8) {

```

```

38         System.out.println("Input " + start + " must be 8 characters long - Exit");
39     }
40     else {
41         String hash = start;                                // declaring a String hash that
42                                                         will hold the final reduced hash of a given password
43
44         // hashing & reducing the start String 10000 times.
45         for (int i = 0; i < 10000; i++) {
46             hash = reductionFunction((hashFunction(hash)), i);
47         }
48
49         // adding the password & its hash value to the rainbowTable HashMap
50         rainbowTable.put(start, hash);
51     }
52     // printing out the contents of the rainbowTable
53     System.out.println(rainbowTable);
54
55     // chain lookup section
56     // long array of the 4 hashes to be searched for
57     long[] hashes = {895210601874431214L, 750105908431234638L, 11111111115664932L,
58                     977984261343652499L};
59
60     // for each loop that loops through each hash in the array of hashes
61     for (long hash : hashes) {
62
63         // looping 10000 times to search for the password - this will function as our max
64         // number of iterations, as 10000 iterations should just take use back to where we
65         // started.
66         for (int i = 0; i < 10000; i++) {
67             // reducing the hash
68             String str = reductionFunction(hash, i);
69             // checking if the reduced hash is a key (password) in the rainbowTable HashMap
70             if (rainbowTable.containsValue(str)) {
71                 System.out.println("Found password " + str + " for hash value " + hash); //
72                 // printing the found password
73                 break; //
74                 // breaking out of the for loop
75             }
76             else {
77                 hash = hashFunction(str); //
78                 // hashing str before continuing the for loop
79             }
80         }
81     }
82 }
83
84 private static long hashFunction(String s){
85     long ret = 0;
86     int i;
87     long[] hashA = new long[]{1, 1, 1, 1};
88
89     String filler, sIn;
90
91     int DIV = 65536;
92
93     filler = new String("ABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGH");
94
95     sIn = s + filler; // Add characters, now have "<input>HABCDEFG..."
96     sIn = sIn.substring(0, 64); // Limit string to first 64 characters
97
98     for (i = 0; i < sIn.length(); i++) {
99         char byPos = sIn.charAt(i); // get i'th character
100         hashA[0] += (byPos * 17111); // Note: A += B means A = A + B
101         hashA[1] += (hashA[0] + byPos * 31349);
102         hashA[2] += (hashA[1] - byPos * 101302);
103         hashA[3] += (byPos * 79001);
104     }
105
106     ret = (hashA[0] + hashA[2]) + (hashA[1] * hashA[3]);
107     if (ret < 0) ret *= -1;
108     return ret;
109 }
110
111 private static String reductionFunction(long val, int round) { // Note that for the first
112     // function call "round" has to be 0,
113     String car, out; // and has to be incremented by
114                     // one with every subsequent call.
115     int i; // I.e. "round" created
116     // variations of the reduction function.
117     char dat;
118
119     car = new String("0123456789ABCDEFGHIJKLMNQRSTUNVXYZabcdefghijklmnopqrstuvwxyz!#");
120     out = new String("");
121
122     for (i = 0; i < 8; i++) {
123         val -= round;
124         dat = (char) (val % 63);
125         val = val / 83;
126         out = out + car.charAt(dat);
127     }
128 }

```

```
120     return out;
121 }
122 }
```

2.2 Output

I couldn't actually find a password match with the above code, and I'm not sure why. My current guess would be that the reduction function wasn't being called properly, as everything else *seemed* to be working as expected. I didn't call the reduction more than 10,000 times as that would theoretically just lead me back to the same place in the chain. I think that my problem is with the passing of the integer *i* to the reduction function, as I think that I correctly implemented the rest of the steps for performing a chain lookup - I input a hash value, reduce it, check if the reduced form is in the list of final plaintexts (the "Values" in the HashMap), and if so break out of the loop (but this never occurs), assigning the relevant "Key" from the HashMap as the original plaintext password that produced the original input hash. Otherwise, I continue until I'm back at the same place in the chain after the 10,000th iteration, where the code gives up.

```
andrew@inspiren3501 CT255-Assignment-2]$ javac RainbowTable.java && java RainbowTable
{Kermit12=lsXcRAuN, GalwayNo=9PZjwF5c, aaaaaaaa=LtVvpQS, HelloPat=dkMPG7!U, Modulus!=L2rEsY8h, Pigtail1=RCNoLfow, pinky##!=eDx58FRq, 01!19!56=vJ90ePjV, Trumpets=!oehRZpK}
andrew@inspiren3501 CT255-Assignment-2]$
```